Mars Pathfinder Flight Software

September 10, 1997

Glenn Reeves
FSW Cognizant Engineer

Outline

- Challenges and Priorities
- Statistics
- Attitude and Information Management Subsystem Description
- Software Development Process
- Software Architecture
- Top 10 Lessons Learned

Challenges and Priorities

- Challenges
  - » Develop the flight software MPF in ~28 months
    - – Three distinct missions : Cruise, EDL, Surface Operations
  - » Define what Better, Cheaper, Faster means for flight software development
- Priorities
  - » Produce flight software that met the requirements of the mission
  - » Produce flight software that contributed to lower operations cost.
  - » Be conscientious of overall cost to the project.

Schedule and Cost

- Schedule
  - » CogE. and 1 engineer started in 5/93 (pre-project)
  - » 10/01/93 Project Start, 11/193 funding begins
    - – All other team members came on board 11/93
  - » 11/15/93 First FSW EEIS demonstration - Uplink, downlink, and rover communications
  - » 3/17/94 First Formal FSW delivery (partial functionality)
    - – Other deliveries 7/94, 3/95, 7/95, 10/95, 1/96, 3/96, 5/96, (and about every month after that)
  - » 6/16/94 FSW PDR, 12/1/94 FSW DDR
  - » Last delivery - last “fixes”, November 1996
- Cost and Workforce
  - » Original $2727K, 14.7 MY
  - » Cost at Completion - $4190K, 23.5 MY

Code Size

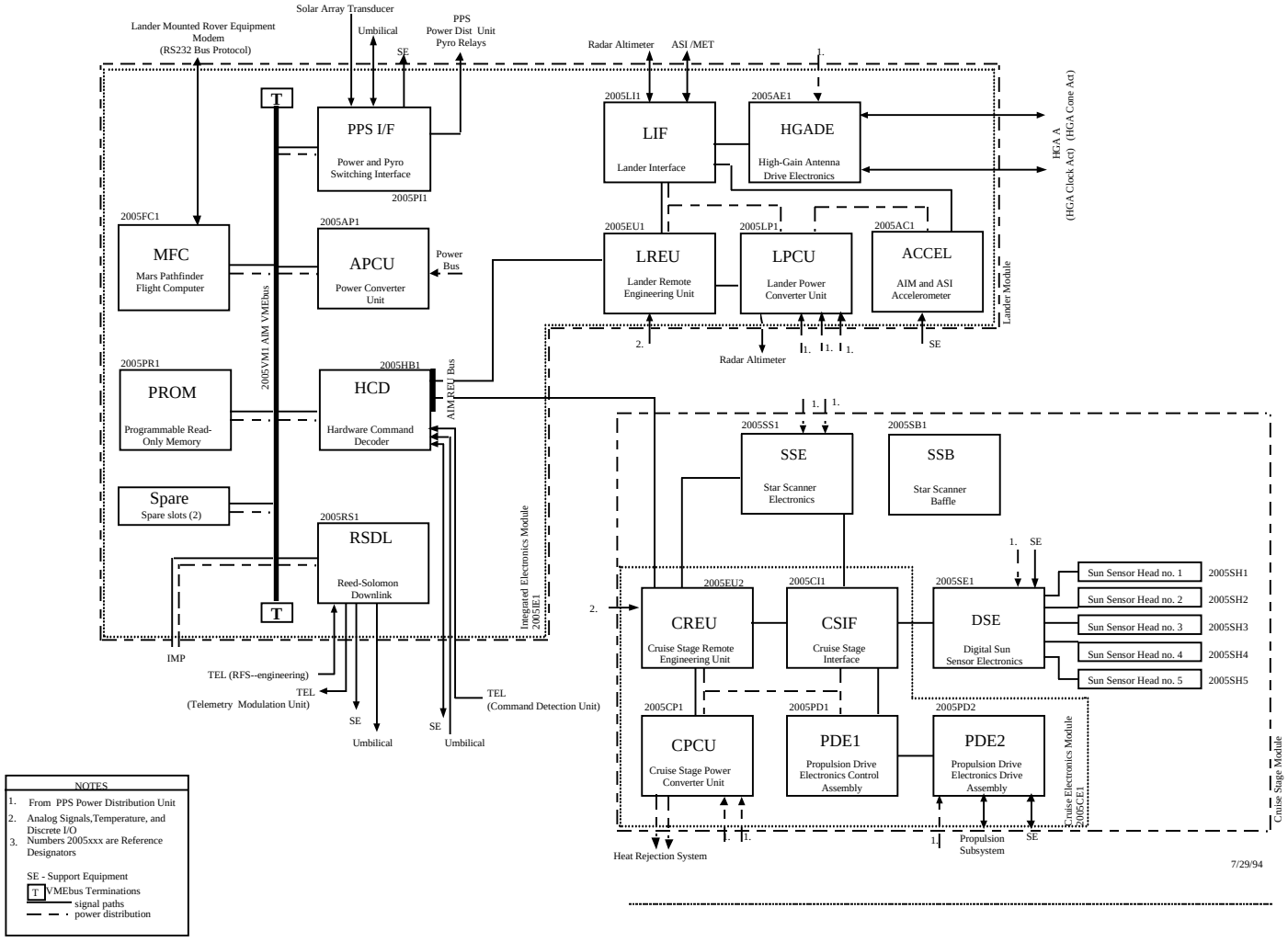
- Code Size - 155K SLOC not including the vxWorks operating system
  - » Original SLOC estimate was 20K SLOC
  - » All software was developed at JPL except IMP which was done by University of Arizona, Tucson and the Rover software which was developed as a separate activity.
  - » 2.1 MB EEPROM storage per copy, two copies on board
  - » ~28 MB RAM used (~15 MB used for image buffers)

Attitude and Information Management Subsystem

- Embedded computing system that manages all high-level spacecraft functions
  - » Accepts commands from Earth
  - » Produces science and engineering telemetry
  - » Attitude Control
  - » Executes complex behaviors
  - » Recognizes and responds to spacecraft fault conditions
- Controls:
  - » Thrusters - Star Scanner
  - » Heaters - Sun Sensor
  - » Valves - Pyrotechnic Devices
  - » Science Experiments
  - » Telecommunications Hardware

AIM Block Diagram

AIM Subsystem Block Diagram



Mars Flight Computer (MFC)

- RAD 6000-SC (Loral Federal Systems)
- Radiation hard, single-chip implementation of IBM RS-6000 architecture
- 22.1 Vax Mips at 20 Mhz
- 128 MB RAM on board
- EDAC protected RAM and data bus

AIM Flight Software Functions

- Command reception and command sequence execution
- Telemetry collection and radio subsystem management
- Attitude determination and control
- Camera control and compression
- Rover communication
- ASI/MET instrument management
- Autonomous Entry, Descent, and Landing
- Autonomous anomaly detection and response
- High Gain Antenna pointing and control
- Power distribution control
- Mission Phase state knowledge
- Flight software load

AIM Flight Software Approach

- C programming language
- Object Oriented Design
- Commercial real-time Operating System (vxWorks)
  - » Priority based scheduling
  - » Memory Management
  - » Intertask communications
- Event driven software architecture
- Rapid prototyping
- Hardware-in-the-loop and software simulation test environments

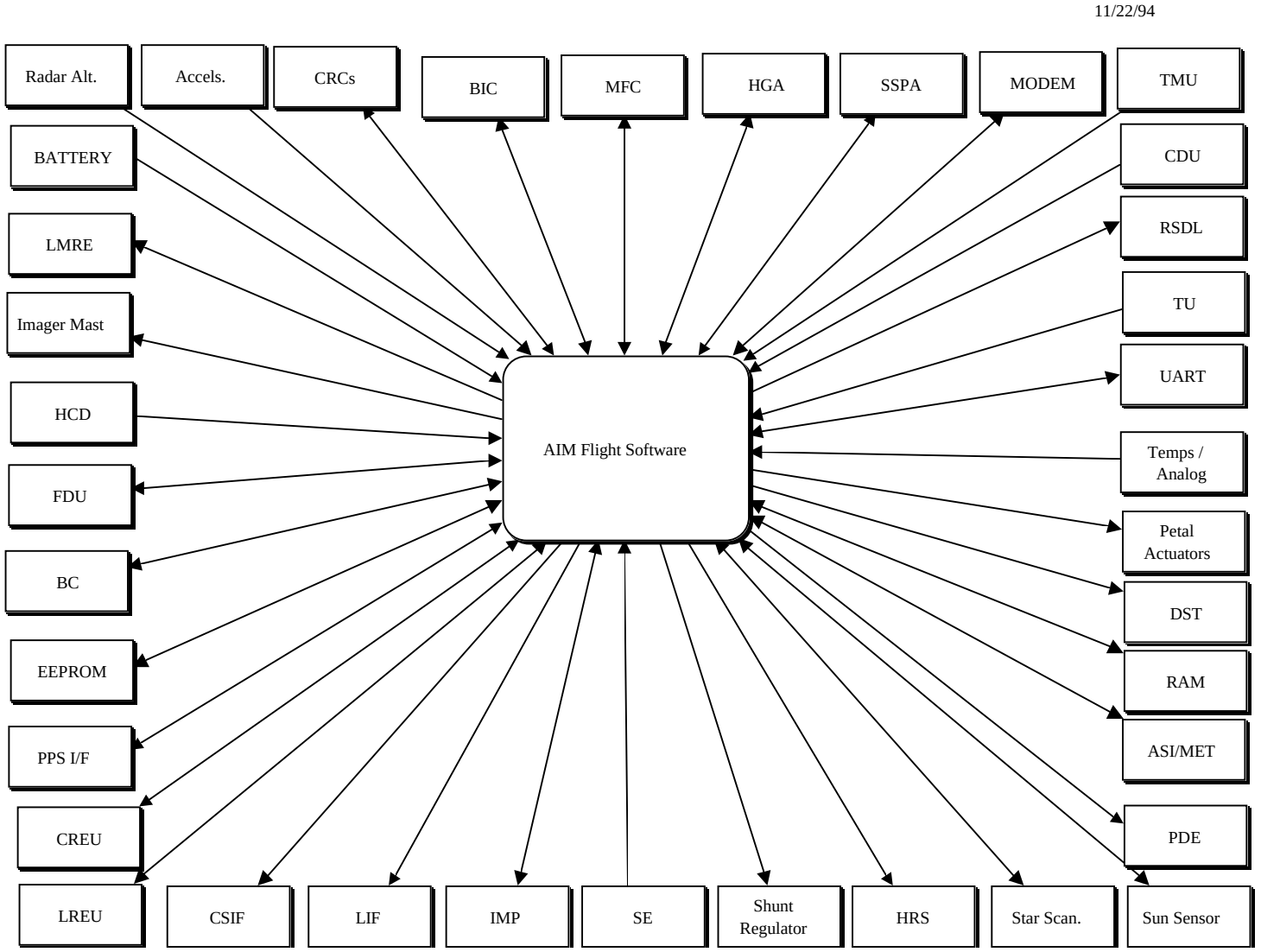
Development Approach

- Object Oriented Design Chosen early in project
  - » The team used a hybrid OOA/OOD with some structured analysis techniques (i,e context diagrams) and functional decomposition to “bracket” the problem.
- First experience with this methodology
  - » Unfortunately, when we started, only one team member had any real experience. For several months the team explored several methodologies including Wirfs-Brock and Rumbaugh.
- Smoke and Mirrors or Sliced Bread?

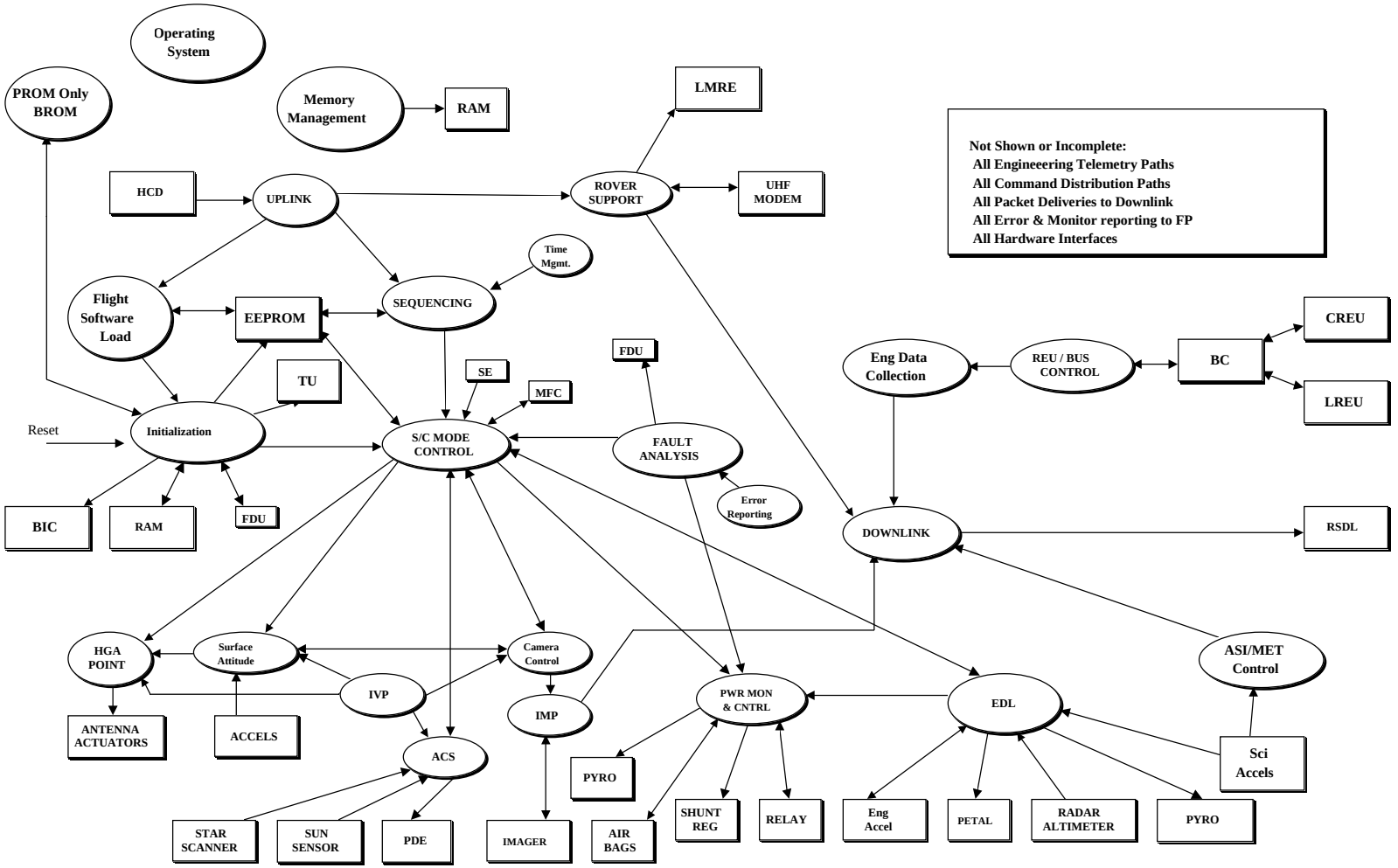
The "Meet-in-the-Middle" Methodology

- Use team experience with spacecraft development and familiarity with mission objectives to develop functional software "subsystems"
- "Top-down" subsystems delegated responsibilities and assigned to individuals for discovery and to develop
- Developers "discover" requirements and responsibilities
  - » Discovery of responsibilities was done by the developer acting like a system engineer
- Developers define objects and collaborations to accomplish subsystem responsibilities
- Complex chains of object collaborations became compact subsystem interfaces
- Objects accepted or rejected by developers based on contribution toward fulfilling subsystem objectives
- This design approach proved extremely effective

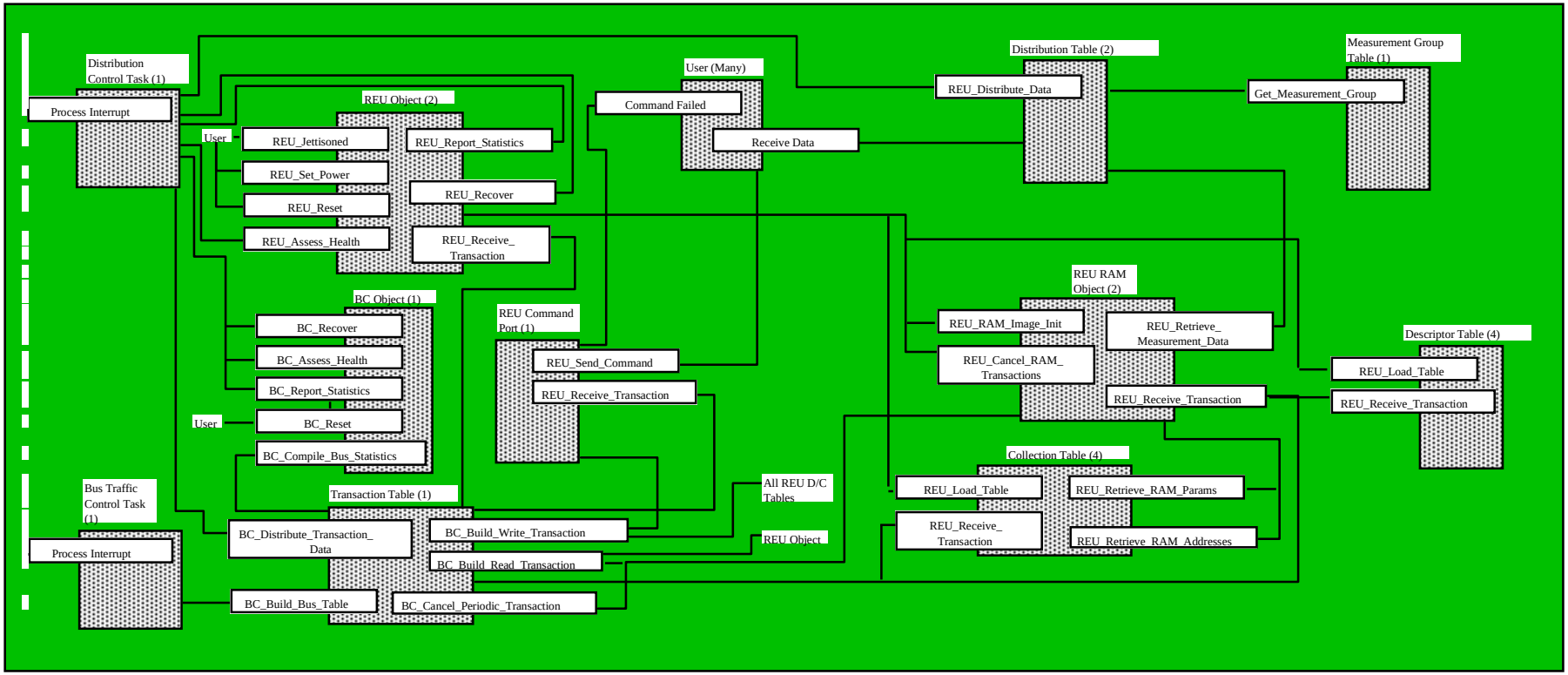
The "Medusa" Diagram



AIM FSW Decomposition



Subsystem Collaboration Diagram Example

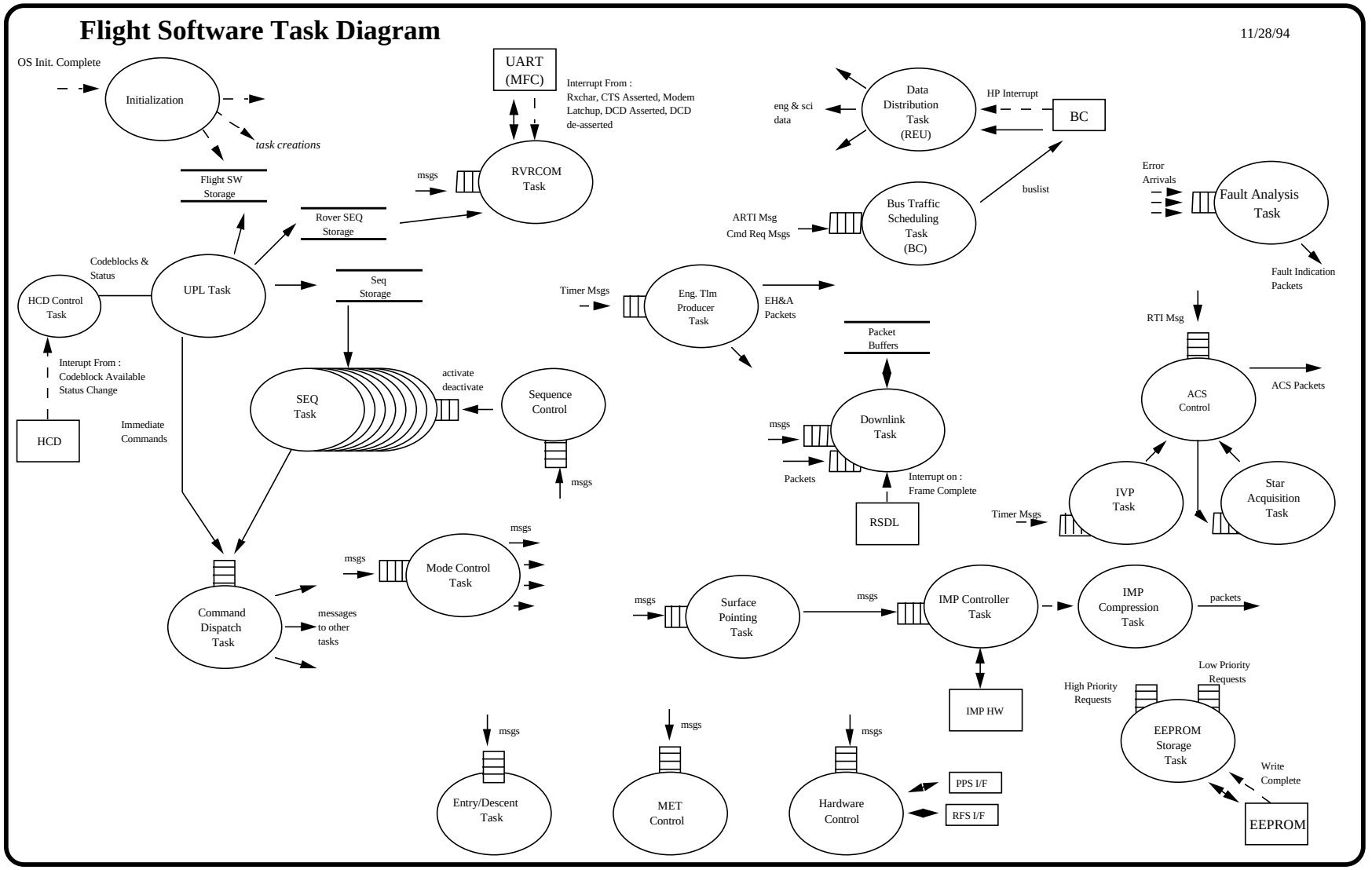


Engineering Bus Subsystem

Architecture

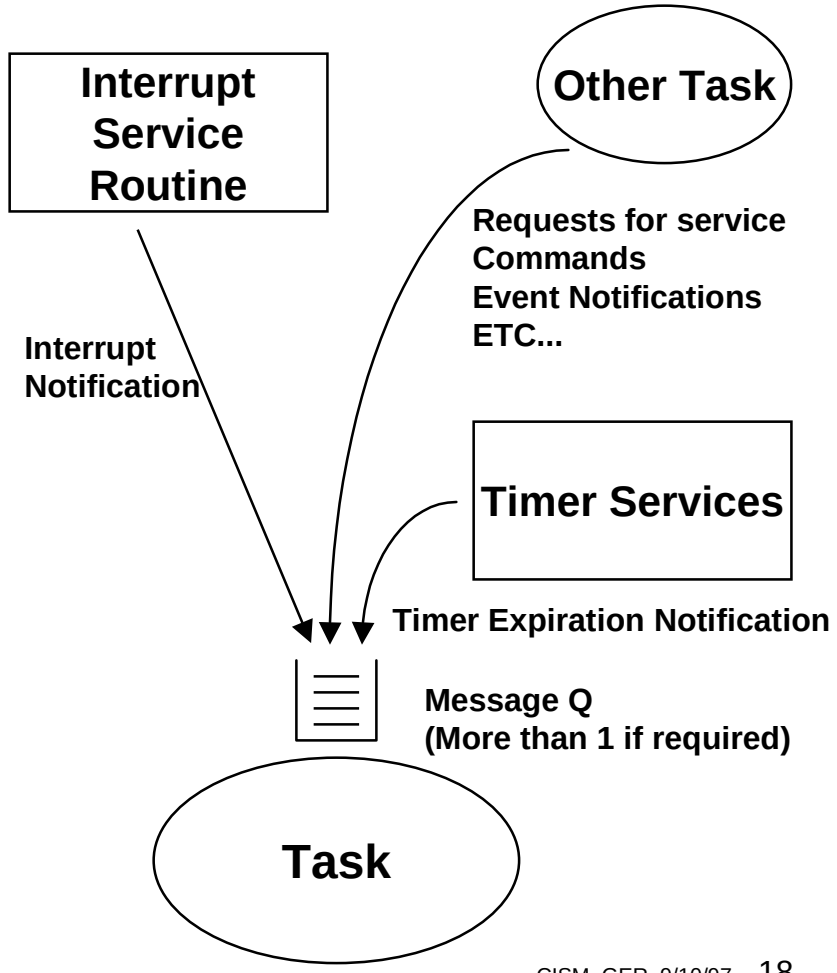
- Multiple task implementation
  - » OS provides context switch actions
  - » Task priority based on task criticality
- Tasks communicate by message passing (primarily)
  - » Implementation based on OS services with wrapper
  - » Multiple input queues permitted per task
    - – Message queue servicing order defines message priority
- Event driven approach
  - » message arrival, interrupt, time expiration
  - » All “events” were delivered as messages
  - » Tasks execute until “work” is complete

FSW Task Diagram



Standard Task Structure

```
BEGIN TASK
    Perform task initialization functions
    DO FOREVER
        Wait for arrival of message
        SWITCH ON MESSAGE TYPE
            WHEN request for service A:
                Handle request
            WHEN request for service B:
                Handle request
            WHEN interrupt notification:
                Handle interrupt
            WHEN timer X expiration:
                Handle timer expiration
            WHEN timer Y expiration:
                Handle timer expiration
        END SWITCH
    END DO
END TASK
```



Top 10 Lessons Learned

- FSW CogE. hand picked all of the FSW team members. They were not assigned to MPF by management. **The makeup of the team is possible the most important element in a successful development. The team lead must have team selection under their control.**
- Each of the flight software team members became a system engineer. They were tasked with discovering the mission, system and subsystem requirements then creating a design, the implementation, and performing the testing on the software they developed. There were very few designs “thrown over the fence” for systems. **Using the software engineers as system designers is efficient; making them discover the requirements gives them perspective.**
- A chief software architect is required to develop the overall software architecture and to make the choice in design and implementation decisions. The architect must also be a strong leader and have the technical skills to understand all aspects of the design. **There must be a guiding individual responsible for the technical aspects of the overall software.** If the chief architect is also the FSW CogE. a programmatic deputy may be required to off load the CogE. This was the approach used successfully on MPF.
- **co-location should be defined as the being able to find the person you need to talk to by yelling their name. It is critical and it is important. It cannot be replaced by email, web pages, or video conferences.** Anyone who believes it can be replaced is wrong. All of the FSW team was located with 60 feet of each other and the lab. The FSW lead was located with the team. The Flight System manager and the AIM subsystem manager were just down the hall next to the chief engineer.

Top 10 Lessons Learned

- **A 20 MIP single board computer with large memory (128 MB) provided flexibility in both flight and ground development and avoided tight margin management.** This choice both permitted and forced the development of the flight software using mature, commercial, workstation based tools. Since these tools do not offer exceptional flexibility in terms of code generation, code placement, and optimization, we were forced to assume the (simplifying) mechanism supported by tools in the workstation environment.
- The Attitude and Information Management (AIM) subsystem electronics used a commercial standard backplane (VME) and an avionics standard serial bus (1553). This allowed the fast development of test environments and platforms using commercial hardware and software. Using mature standards practically guaranteed that problems detected by the commercial equipment were problems in the flight hardware or software and not vice versa. **It was critical to the schedule to use mature interface specifications such that (relatively) inexpensive mature commercial test hardware could be used. All of the test environments, including the S/C, benefited from this selection.**
- The FSW is the user of the hardware and the FSW team is the customer. **The FSW team should levy requirements on the hardware and review the hardware designs prior to the hardware being built. FSW representation should exist on the hardware design team.**

Top 10 Lessons Learned

- **A "Flight Like Test Set" was developed with commercial hardware and simulation software built by the flight software team members.** This testbed was used for unit testing and early integration testing. Since it used vxWorks no flight software changes were required; only re-compilation. An definite advantage of the VME architecture for flight was that the same memory mapped I/O could be used in the FLTS. The flight software was implemented so that a separate "driver" layer separated the interaction with the hardware from the application software. By changing the addresses used by the driver layer and re-compiling, the FLTS was accommodated.
- **Get software and hardware together early in development cycle.**
- The flight software team did all of the hardware software integration activities. In general, there was no separate integration effort outside of the flight software team. **Let the FSW team be the integrators of the hardware and the software. In the end, a third party T&I team, cannot fix the problems; the FSW team fixes the problems so they might as well discover them too.** The T&I team should really emphasize the Test aspect.
- **A dedicated flight software testbed with engineering model hardware was "owned" by the flight software team. The benefit of this testbed was enormous. All of the VME based hardware was included. A complete testbed with all of the ACS sensors and electronics would have allowed parallel development and checkout of the ACS software.**

Conclusion

- A very successful mission
- A successful flight software development and a good product
  - » A healthy set of lessons to learn from too
- MPF flight software is now being used within several other JPL and outside projects
  - » SeaWinds, Mars98, SIRTF, DS-1
- Several of the innovations from MPF FSW will be used in X2000